Analysis of Gene Expression Data **Spring Semester, 2002**

Lecture 6: March 31, 2005

Lecturer: Gideon Dror *Scribe: Daniela Raijman and Igor Ulitsky* [1]

6.1 Introduction to Classification

One of the major current applications of microarray technology is using the genome-wide expression data in order to classify samples taken from different tissues. Classifying cancer tissues for diagnosis purposes is already being used in the Netherlands, and many other countries are expected to introduce this technology into the medical practice in the near future.
Classification as a discipline is usually viewed in the context of *Machine Learning* [3], a form of *Artificial Intelligence*. Classification is a form of *Supervised Learning*, which is sometimes termed as "learning with a teacher". The algorithm is first presented with an initial *training set* and is expected to extract from it sufficient information in order to successfully handle never-seen inputs. Other common problems in supervised learning are *Ranking* and *Regression*.

6.2 Problem Definition

For simplicity, this lecture will deal solely with *binary classification* :

Definition The problem of binary classification is defined as:
Input: a set of m examples (x^j, y^j) $j = 1, 2...m$ (the *learning set*) sampled from some distribution D, where $x^j \in R^n$ and $y^j \in -1, +1$. The i-th component of x^j - x_i^j is termed *feature* i.
Output: a function $f : R^n \rightarrow -1, +1$ which classifies "well" additional samples x^k sampled from the same distribution D.

In the rest of the scribe X_i will denote the i-th *feature* and x_i^j the sample corresponding to that feature.

6.3 Example Applications

Example *Classification of tissue samples using gene expression data.* In this case each measured gene comprises a feature and the learning set is composed of vectors containing gene expression measurements for different tissues. The problem can be to classify the tissues as malignant or healthy (Y = malignant/healthy) or to distinguish between different types

[1] Based in part on a scribe by Simon Kamenkovich and Erez Greenstein May 2002.

of cancer. An example of the use of classification with gene expression data can be seen in Figure 6.1.

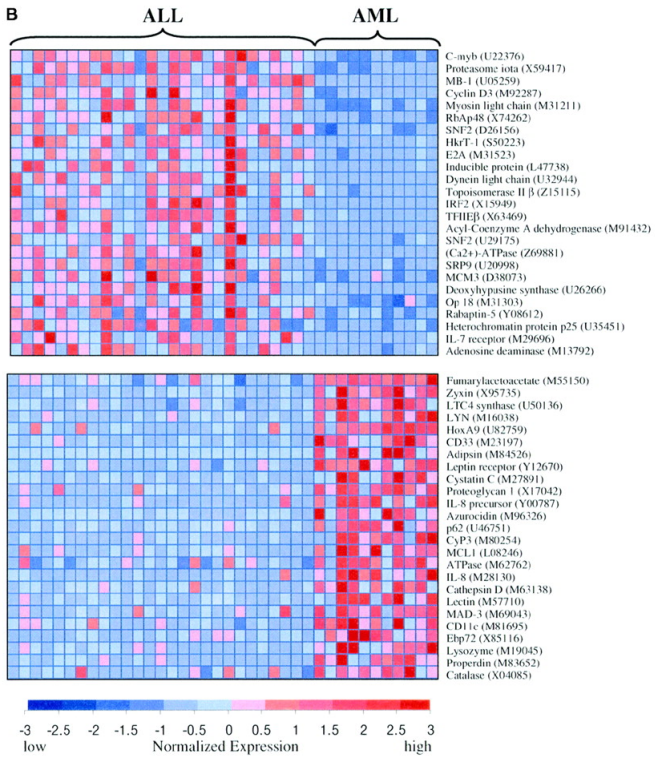

Figure 6.1: Genes distringuishing ALL from AML (two types of leukemia). The 50 genes which give the most information regarding the ALL-AML class are shown. Each row corresponds to a gene, with the columns correspinding to expression level in different samples. Expression levels for each gene are normalized across the samples such that the mean is 0 and the standard deviation is 1. Normalized expression levels greater than the mean are shaded in red, and those below the mean are shaded in blue. The scale indicates SDs above or below the mean. .

Example *Detection of spam mail.* In this case the X is some vector representation of e-mail messages (e.g. X_i - the number of times the i-th word from some fixed dictionary appears in the message). The problem is to classify the mail into spam and non-spam. The training set in this case is list of messages that have been manually classified.

Example *Face Detection* - the problem of deciding whether a given image (represented as a vector of pixels) represents a human face. In this case X_i can be the color intensity of the ith pixel.

Example *Signature Recognition* - the classifier can be trained to recognize whether a certain signature belongs to a certain person.

The input to a classification problem can be visualized as in 6.2-A as a set of points in the plain, colored with 2 colors representing the two classes that need to be separated.

6.4 Classification Algorithms

6.4.1 Classification Function

All *Classification Algorithms* (termed *classifiers*) can be generalized as algorithms that receive a training set and learn a *classification function* of the form $f : R^n \to \{+1, -1\}$. This function is then applied to new inputs and its value represents the class to which the input is classified. Thus, different classification algorithms differ in the form of function they are capable of learning. The common classification algorithms include: Fisher Linear Discriminant (6.4.3), KNN (6.4.4), Decision Trees (6.4.5), Neural networks (6.4.6), Naive Bayes (6.4.7), SVM (6.7), and Adaboost ([6]).

6.4.2 Properties of a Classifier

Training speed The amount of time the classifier needs to learn the classification function given a training set of a certain size.

Accuracy The accuracy of a classifier can be evaluated using a *test set* with known class derivation for every item. The classification error can be tested using varying schemes, for instance bearing different penalties for false positive and false negative mistakes.

Transparency Some classification functions are very clear (for example a threshold function), making it possible to derive important insights about the problem or the data. On the other hand, some functions are very complex, making any such conclusions infeasible. A common example for a classification function which lacks clarity transparency is the classification performed using *Neural Networks*.

6.4.3 Fisher Linear Discriminant

The *Fisher Linear Discriminant* [4] method of classification finds a direction w (a vector in R^n) in the n-dimensional space.
Given a sample which needs to be classifier, the Fisher classifier calculates the projection of the sample onto the direction. The idea is to find a direction which after the projection will *maximize interclass variability and minimize intraclass variability*. In the simple two-dimensional case, after the points are projected onto the line, the two classes are transformed into two sets of points upon the line. The *interclass variability* in this case is the distance between the class centers, and the *intraclass variability* is the distance of class members from their class center.

Different criteria can be employed to determine the class for a new sample. For instance:
• Calculating the distance from the point to the means for the projections of the training classes.
• As above, but adding a weighting scheme in order to minimize the bias caused by the

relative sizes of the training classes.
The advantage of Fisher linear discriminant scheme is that such a vector can be found swiftly using a simple procedure.

6.4.4 k Nearest Neighbors

The *k nearest neighbors (KNN)* classification scheme employes a more local method of classification. The method requires no initial processing of the training data, and new samples are classified based on its k nearest neighbors in the training set. The KNN classifier has numerous variants, as the concept of proximity can be defined in various manners, as well as the decision rule of the new sample class based on the neighboring samples. For example, a simple variant of KNN would find the neighbors based on euclidian distance and use the majority rule to set the classification of the new sample. The KNN scheme is depicted in Figure 6.2-B. This kind of classifier is able to learn a relatively complex separation function. A drawback of this method is that in some practical problems, the euclidian distance is inappropriate, and the "correct" distance metric is difficult to define. Another problem is that as the method performs no preprocessing of the sample, the major computational complexity occurs while classifying unseen samples, a stage which should usually be performed swiftly.

6.4.5 Decision Tree

The *Decision Tree* method constructs a tree representing the classification process. The leaves of the tree represent one of the classes and the internal nodes contain some sort of desicion function of the input sample with a boolean value. In the simplest case, this function is a predicate on one of the feathers, e.g. $x_1 > 3$ or $x_5 < 7$. The tree is usually constructed by selecting the most informative feature X_i at every step and constructing an internal node in the tree discriminating based on this feature. A sample decision tree is presented in Figure 6.2-C. One of the advantages of the decision tree model is the relative classification speed.

6.4.6 Neural Networks

A neural network [2] is a combination of inter-connected network where the nodes correspond to *neurons* and the arcs correspond to synaptic connections in the biological metaphor. The knowledge of the network is stored in the arc weights. A simple neural network with one layer and a single output neuron is termed *Perceptron* and is capable of distinguishing between classes which can be separated by a straight line (hyperplane) (as shown in Figure 6.2-D). In this the perceptron is somewhat similar to the fisher discriminant classifier. More complex neural networks with multiple layers and multiple output neurons are theoretically capable of separation using any continuous surface. However, the neural network model suffers from several drawbacks:
(1) The function constructed by the neural network lacks transparency, making it impossible to deduce conclusions regarding the data. In other words, the neural network is a "black box", which performs well in some situations. (2) The iterative algorithm employed for learning the classification function may converge slowly or not at all for some problems. (3)

As any gradient-based iterative optimization search algorithm, the learning of the neural networks is susceptible to local minima.

6.4.7 Naïve bayes

The naive bayes classifier is based on the concepts of *Bayesian decision theory*, which is closely related to *hypothesis testing*. The two classes are treated as the two hypotheses : A is "this sample belongs to class A" and B is "this sample belongs to class B". In Bayes theory, in order to decide among the two hypotheses the *log-likelihood ratio* is computed - $LR(x^j) = \log \frac{L_A(x^j)}{L_B(x^j)}$. If $LR(Data) > \log \frac{1-\lambda}{\lambda}$, x^j is classified to class A. λ in this case is the *prior* probability that x^j belongs to class A. In this case it is assumed that the costs of deciding A when B is correct and vise versa are identical. For a detailed explanation of the naive bayes method see [5].

6.5 Dimensionality Reduction

Up to this point in the lecture we have dealt solely with n-dimensional data, n being the original dimension of the input data. A reduction of the data dimension can carry with it several important advantages for both learning and classification.

6.5.1 Dealing with Overfitting

One of the major problems encountered by all classifying schemes is *overfitting* of the data. The data in the learning set can be viewed as containing *general* information characterizing the population, along with information specific to the sampled training set. An ideal classifier is supposed to work only on the general parameters. This is usually termed as performing *generalization*. If the classifier adheres strongly to signals specific to the learning set it is said to *overfit* it. For example, in the decision tree classifying scheme, a large tree containing multiple complex functions with a single training sample at each "leaf", will probably perform superbly on the training set, but perform poorly on new samples.
Overall, any complex separating function is vulnerable to overfitting, as can be seen in Figure 6.2-E. Reducing the dimensionality of the data can usually help overcome the maladies of overfitting by allowing the classifier to focus on the important features.

6.5.2 Merits of the Dimensionality Reduction

• As described above, overfitting is reduced.
• A well-performed dimensionality reduction can improve the performance of the classifier, by removing features which do not contribute to the classification, and may circumvent it with misleading noise.
• Several of the classification algorithms suffer from difficulties when dealing with high-dimensional data.
• In all classification schemes, high dimension causes greater time/memory consumption in the learning and classification phases.

• The use of fewer dimensions improves the clarity of the classification, allowing a better understanding of the meaningful signals found in the data.
• In the context of gene expression data, it is significantly easier, cheaper and more accurate to deal with expression measures from a small number of genes (e.g. 50) instead of a whole-genome survey (including up to 40,000 probes).

6.5.3 Approaches to Dimensionality Reduction

Feature Construction

In the *Feature Construction* approach, n features of the input are transformed into l other features using some linear/non-linear transformation (e.g. rotation). For example, in the application to face recognition problem, the n pixels extracted from the image can be reduced to a set of distances between points with distinctive colors. A common method in feature construction is *Principal Component Analysis* - PCA [4], an analytical method with finds a linear transformation that chooses a new coordinate system for the data set. In the new coordinates, the greatest variance by any projection of the data set comes to lie on the first axis (termed the first principal component), the second greatest variance on the second axis, and so on.

Feature Selection

In *Feature Selection*, given a training set of n dimensional samples x^j we're interested in selecting l features which maximize some trait we're interested in. This trait can of course be the performance of the classification process, but can also be something else, for instance - detecting the important features (e.g. genes) in the training set. In this scribe we'll focus on maximizing the prediction accuracy, meaning that given a new sample x^k from the same distribution D as the training data we'll minimize:

$$\sum_j [f(x^k) - y_j]^2 \text{ or } E(|f(x^k) - y_j|)$$

Selecting the features

A exhaustive search among the possible sets of selected features is infeasible in almost all practical cases, so heuristics are commonly employed. The integration of those with the classifier can be divided into three cases, which are depicted in Figure 6.2-F :

Filter The features are selected in a separate process before the classifier is trained.

Wrapper In an iterative manner, the learning process alternates between selecting features and learning a classification function, improving the feature selection process based on the feedback from the classifier. Various optimization techniques can be used here, for example *hill climbing*.

Embedded The selection process is embedded in the classifier. The difference from the Wrapper scheme is that in this case the two processes cannot be separated into two iteration phases. For example, the learning process of the decision trees includes an implicit selection of the features that appear in the node functions.

As can be seen from the definition, while in the Filter method the classifier is ignored in the selection method, Wrapper and Embedded techniques can accommodate themselves to maximize the performance of a specific classifier.

6.5.4 Filtering of the Features

Even though the filtering approach does not take into the consideration properties specific to the classifier, it can accomplish suprizingly good results. The filtering is usually based on extracting features X_i which are more *informative* about the the class Y. This notion of information can be captured by several means:

- *Pearson correlation* $\rho(X_i) = \frac{cov(X_i,Y)}{\sigma(X_i)\cdot\sigma(Y)}$: Using this measure features which are highly correlative to the class will be extracted. An example of such a feature can be seen in Figure 6.2-G.

- χ^2 : This measure will extract features whose *distribution* is similar to that of Y.

- *Fisher criterion* $F(X_i) = \frac{\mu^+_{X_i} - \mu^-_{X_i}}{\sqrt{{\sigma^+_{X_i}}^2 + {\sigma^-_{X_i}}^2}} > C$: This measure prefers features with distinctively different distributions between the two target classes.

- *Golub criterion* $F(X_i) = \frac{\mu^+_{X_i} - \mu^-_{X_i}}{\|{\sigma^+_{X_i}}^2 + {\sigma^-_{X_i}}^2\|} > C$: Similar to Fisher criterion.

- *Mutual information* $I(X_i, Y) = \sum P(X_i, Y) \log \frac{P(X_i,Y)}{P(X_i)P(Y)}$: A mesure derived from *Information Theory*.

For all the measures described about, the most prominent k features are extracted from the original n features. The filtering method suffers from a few drawbacks:

- The filtering performs well when the samples are not correlated. A classic example of an ill performance of filtering is the XOR function.

- As mentioned above, the filtering disregards the classifier.

Nevertheless, the filtering method is very fast, thus allowing multiple filtering schemes to be efficiently tested in order to select the one giving the best performance results. For many practical purposes, filtering performs well, as can be seen in Figure 6.2-H.

6.6 Performance Assessment

A crucial part of building a classifier lies in evaluating its performance. This evaluation is very important, as the classifier usually employs several parameters whose optimal values can be found using the assessment process. Also, in most problems several classifiers are available and the optimal choice is selected through trial and evaluation. In the context of feature selection, the number of features selected is usually determined using performance assessment.

6.6.1 Test Set Estimation

A naive and wrong approach to performance assessment is to use the same data set for training the classifier and assessing its performance. The grave problem in this is that it introduces a downward bias. After a complex enough learning procedure many classifiers are capable to perfectly classify the training data, yet perform poorly on new data (a symptom of overfitting). In order to overcome this, the classifier is trained using some part of the learning set, termed *training set*, and its performance is evaluated using an independent *test set*. The construction of those two sets can be performed using two main approaches:
• The initial learning set is divided into 2 distinct groups - L_1 (training set) and L_2 (test set). In order for this method to perform well, L_1 and L_2 must be approximately independently distributed. The main drawback of this method is the fact that the effective size of the learning set - the number of samples used for training is reduced, thus harming the training process.
• The method of *m-fold cross validation* randomly divides the learning set into m distinct portions of nearly the same size. The learning is composed of m iterations, where in each iteration one portion from the learning set is put aside, and the training is based on the rest of the samples. The performance is then evaluated based on the samples set aside. After m iterations the average performance is calculated. The m parameter value is set considering the bias-variance tradeoff. A larger value for m reduces the bias, but includes the variance of the performance assessment. A special case of the m-fold cross validation occurs when $m = 1$ and it is called *Leave-one-out cross validation (LOOCV)*.
When feature selection is incorporated in the classification (Embedded scheme), the feature selection must be performed only on L_1, in order to avoid a downward bias.
The use training set and test set is often use for detection of overfitting, by plotting the classification error on both sets, as in Figure 6.2-I. The learning process in this case is set to stop at the point where the classification error on the test set starts to increase.

6.7 Support Vector Machines

6.7.1 Introduction

The theory of support vector machines (**SVM**), has its origins in the late seventies, in the work of Vapnik [7] on the theory of statistical learning. Lately it has been receiving increasing attention. Many applications, as well as important theoretical results, are based on this theory. In fact, Support Vector Machines are arguably the most important discovery in the

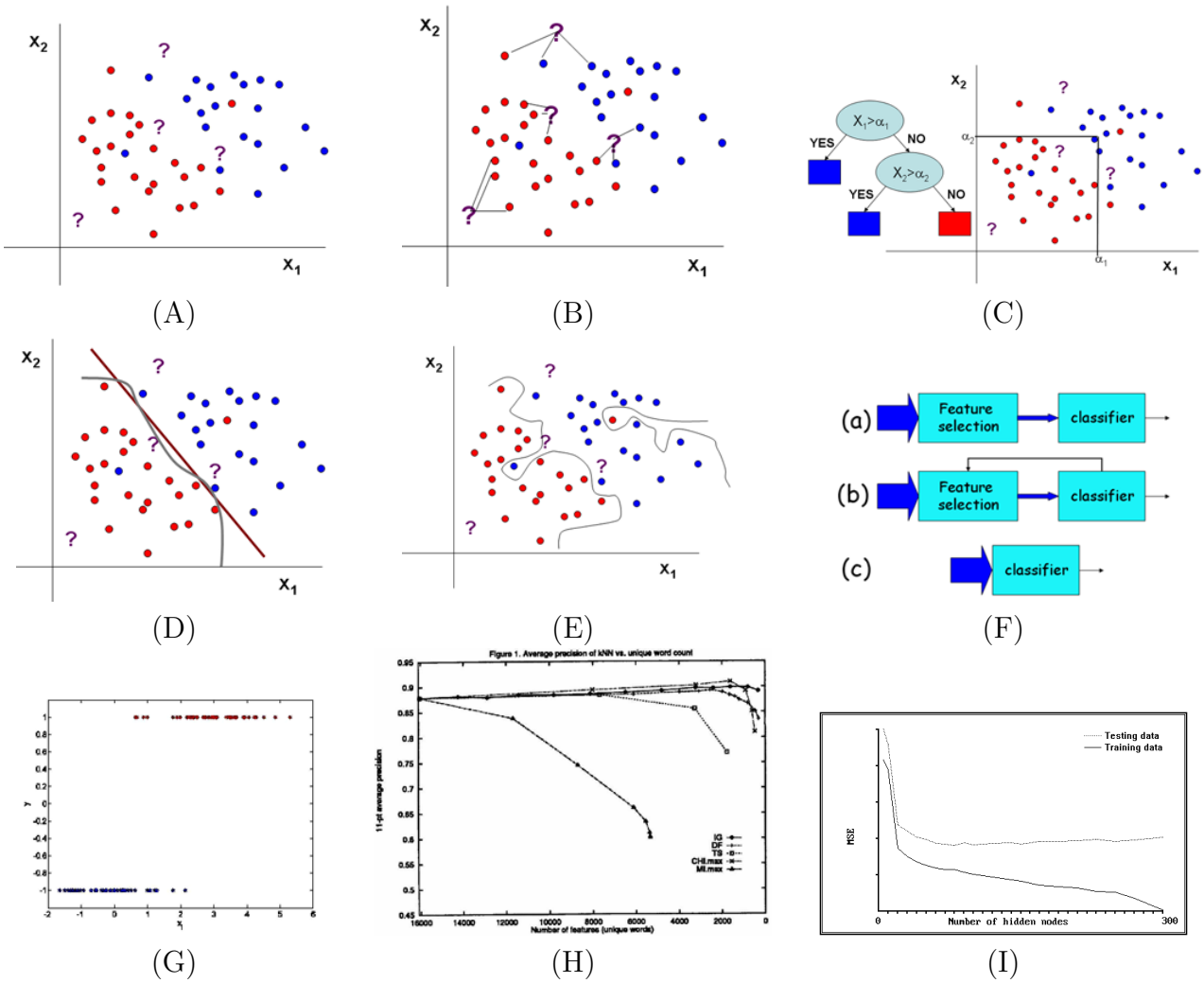

Figure 6.2: (A) Classification in 2D plane. The colored points indicates the samples contained in the training set of the classifier and the "?" the locations of the new samples that need to be classified. (B) Classification using KNN. The class for each new sample (represented by "?") is set based on its k (3 in this case) neighbors. (C) Classification using a decision tree. The tree presented in the left half of the figure describes the decision process, which in this case contains two simple predicates on the features X_1 and X_2. The figure on the right depicts the separation encoded in the tree. (D) Separation by neural networks. The straight line depicts the separation achieved by a simple *Perceptron* and the curve the separation by a multi-layered network, which is in theory able to learn any separating function. (E) A classification function which overfits the training set. (F) The three basic options of introducing feature selection to a classifier. (a) Filter (b) Wrapper (c) Embedded. (G) An example of a classification problem where the feature X_i will be selected using *Pearson correlation*, as the feature is highly correlated with Y. (H) Results of applying the filtering feature selection on a dataset of e-mail messages classified to span/non-spam. The classifier in this case is KNN and the percision of the classifier is plotted agains the number of features used (in this case every unique word in the word count vector is a feature). The graphs represent different measures for filtering: **CHI.max** - χ^2 measure, **MI.max** - Mutual information. As it can be seen, the optimal filtering is performed using χ^2 selecting about 2000 features. (I) The graph above shows the how MSE (error function) changes as a function of the neural network size. The test error for the training data grows smaller. The error for the test data decreases up to a certain point, where it starts to increase. Beyond this point the network suffers from overfitting and does not generalize well.

area of machine learning. The main idea of Support Vector Machines is to find a decision surface - a hyperplane in feature space (a line in the case of two features) - which separates the data into two classes. SVMs extremely successful, robust, efficient, and versatile. In addition, there are good theoretical indications as to why they generalize well. Several efficient implementation methods exist, however these will not be discussed in this scribe. Reference

[1] provides an extensive review on Support Vector Machines. We will discuss the use of support vector machines in three different settings:
- Data are linearly separable
- Data are linearly non-separable
- Data are non-linearly separable

6.7.2 Linear Separable Case

We will start with the simplest case - linear machines trained on separable data. Given a training set $x_i, y_i, x_i \in \mathbf{R}^n, y_i \in \{-1, 1\}$, we assume that the data is linearly separable, i.e., there exists a separating hyperplane which separates the positive examples ($y_i = 1$) from the negative ones ($y_i = -1$). The points x which lie on the hyperplane satisfy $w \cdot x + b = 0$, where w is a normal to the hyperplane and $\frac{|b|}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin. Let $d_+(d_-)$ be the shortest distance from the separating hyperplane to the closest positive (negative) example. Define the *margin* of a separating hyperplane to be $d_+ + d_-$. For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with largest margin.

The goal is to find the optimal linear *classifier* (a hyperplane), such that it classifies every training example correctly, and maximizes the classification margin (See figure 6.3-A). The above description can be formulated in the following way: If class 1 corresponds to 1 and class 2 corresponds to -1, formulated as:

$$x_i \cdot w + b \geq +1, \forall x_i \text{ where } y_i = +1 \tag{6.1}$$
$$x_i \cdot w + b \leq -1, \forall x_i \text{ where } y_i = -1 \tag{6.2}$$

then we can re-write (6.1) and (6.2) as:

$$y_i(w \cdot x_i + b) \geq 1, \forall x_i \tag{6.3}$$

Now consider the points for which the equality in (6.1) holds. These points lie on the hyperplane $H_1 : x_i \cdot w + b - 1 = 0$. Similarly, the points for which the equality in (6.2) holds lie on the hyperplane $H_2 : x_i \cdot w + b + 1 = 0$. These points are called *support vectors* (See figure 6.3-B). In this case $d_+ = d_- = \frac{k}{\|w\|}$, and thus the margin is $\frac{(b+k)-(b-k)}{\|w\|} = \frac{2k}{\|w\|}$. The problem can be rescaled to $k = 1$ for simplicity. Note that H_1 and H_2 are parallel (they have the same normal) and that no training points fall between them. Thus we can find the pair of hyperplanes which gives the maximum margin by maximizing $\frac{2}{\|w\|}$, which is equivalent to minimizing $\|w\|^2$, subject to the above constraints. This is formalized as:

$$minimize \qquad \|w\|^2 \tag{6.4}$$
$$s.t. \quad y_i(w \cdot x_i + b) \geq 1, \forall x_i \tag{6.5}$$

The above minimization problem is convex, therefore there exists a unique global minimum value, and there is a unique minimizer, i.e. weight and b value that provides the

minimum (given that the data is indeed linearly separable). This problem can be efficiently solved using quadratic programming with modern constraint optimization engines.
To summarize our discussion so far, our goal is to find an optimal linear classifier (a hyperplane) such that:
1. It classifies every training example correctly.
2. It maximizes the classification margin.

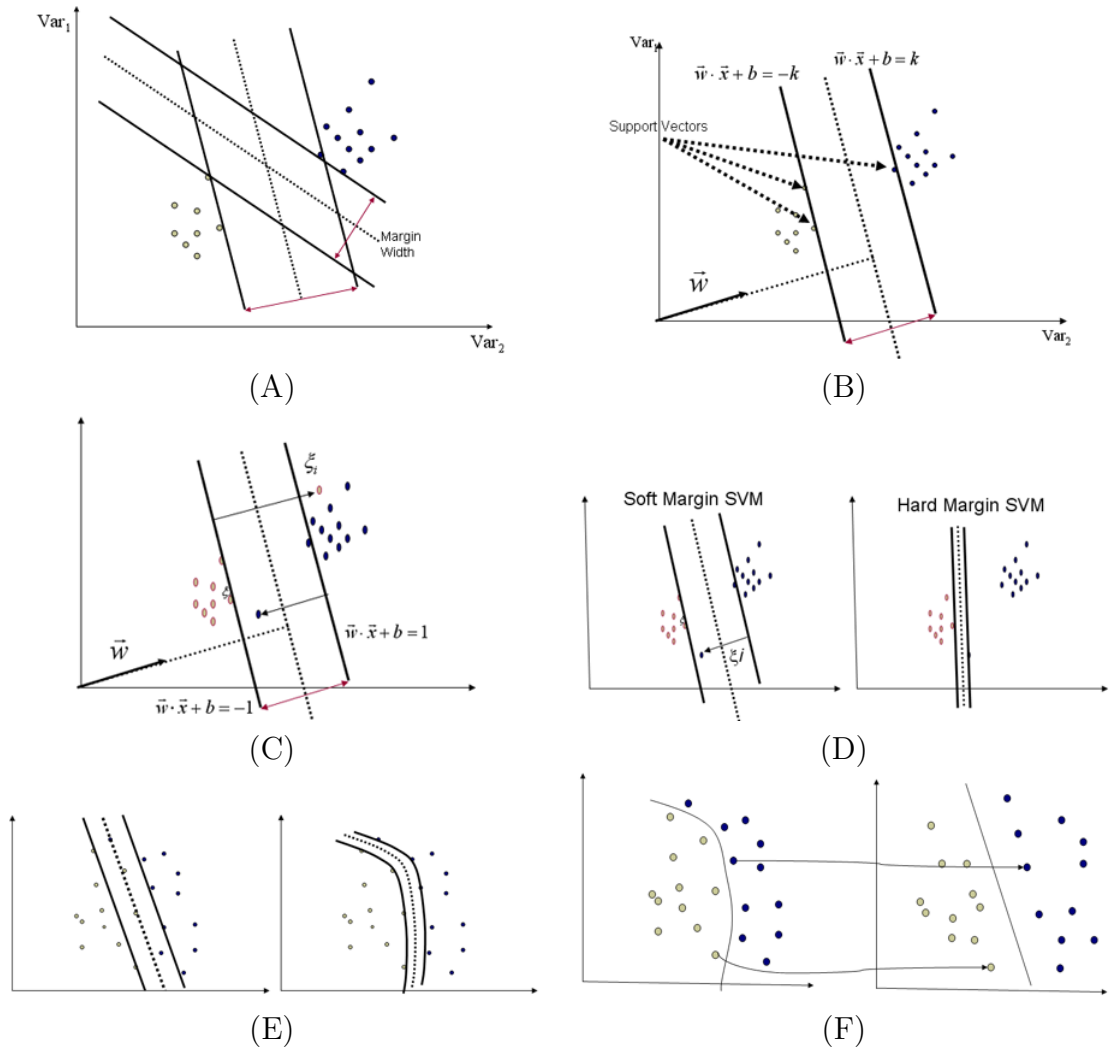

Figure 6.3: (A) There are many options for choosing $w \cdot x + b$. The SVM method chooses the option which maximizes the margin. (B) Choice of w and b which maximizes the margin. Data points lying on H1 and H2 are the support vectors. In this case $d_+ = d_- = \frac{k}{\|w\|}$. (C) Linear separation with outliers. (D) Using a soft margin allows separation may improve margin width even when perfect separation is possible. (E) Non-Linear Separation. (F)Data points can be mapped into a new feature space, such that they are linearly separable in the new space.

Lagrange formulation of the problem

We will now switch to a *Lagrangian formulation* of the problem. There are two reasons for doing this. The first is that the constraints (6.1),(6.2) will be replaced by constraints on the Lagrange multipliers, which will be much easier to handle. The second is that in this

reformulation of the problem, the training data will only appear (in the actual training and test algorithms) in the form of dot products between vectors. This is a crucial property which will allow us to generalize the procedure to the nonlinear case.

Thus, utilizing the theory of Lagrange multipliers, we introduce positive Lagrange multipliers $\alpha_i,\ i = 1, \ldots, l$ one for each of the inequality constraints . We form the Lagrangian:

$$L_P = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} \alpha_i$$

We have to minimize L_P with respect to w, b and simultaneously require that the derivatives of L_P with respect to all the α_i vanish. This is a convex quadratic problem, which has a dual formulation: maximize L_P, subject to the constraint that the gradients of L_P with respect to w and b vanish, and subject to the constraints that $\alpha_i \geq 0$. This is formulated as:

$$aximize \qquad L_D = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \tag{6.6}$$

$$s.t. \qquad \sum_{i=1}^{l} \alpha_i y_i = 0, \alpha_i \geq 0 \tag{6.7}$$

Support vector training (for the separable, linear case) therefore amounts to maximizing L_D with respect to the α_i, subject to the above constraints, with solution given by $w = \sum_{i=1}^{l} \alpha_i y_i x_i$. Notice that there is a Lagrange multiplier α_i for every training point. In the solution, those points for which $\alpha_i > 0$ are the *support vectors*, which lie on one of the hyperplanes H_1 or H_2. All other training points have $\alpha_i = 0$ and lie to the side of H_1 or H_2 such strict inequality holds. For these machines, the support vectors are the critical elements of the training set. They lie closest to the decision boundary. If all other training points were removed (or moved around without crossing or moving too near to H_1 or H_2), and training was repeated, the same separating hyperplane would be found.

6.7.3 Linear Non Separable case

When applied to non-separable data, the above algorithm will find no feasible solution. This will be manifested by the objective function (i.e. the dual Lagrangian) growing arbitrarily large. So how can we extend these ideas to handle non-separable data? We would like to allow some instances to fall within the margin or to be misclassified, but penalize them. That is, we would like to introduce an additional cost (i.e. an increase in the primal objective function) for doing so. This can be done by introducing positive slack variables $\xi_i, i = 1, \ldots, l$ in the constraints (Cortes and Vapnik, 1995), which then become:

$$x_i \cdot w + b \geq +1 - \xi_i,\ y_i = +1 \tag{6.8}$$

$$x_i \cdot w + b \leq -1 + \xi_i,\ y_i = -1 \tag{6.9}$$

Thus, for an error to occur, the corresponding ξ_i must exceed unity, so $\sum_i \xi_i$ is an upper bound on the number of training errors. Hence a natural way to assign an extra cost for errors is to change the objective function to be minimized from $\frac{1}{2}\|w\|^2$ to $\frac{1}{2}\|w\|^2 + C\sum_i^l \xi_i$,

where C is a parameter to be chosen by the user. A larger C corresponds to assigning a higher penalty to errors. This is again a convex quadratic programming problem. Thus the Lagrangian formulation becomes:

$$maximize \qquad L_D = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \tag{6.10}$$

$$s.t \qquad 0 \leq \alpha_i \leq C, \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{6.11}$$

The only difference from the optimal hyperplane case is that the α_i have an upper bound of C. The parameter C controls the range of the α_i and avoids over emphasizing some examples. C is called the complementary *slackness*. When $C \to \infty$ the problem becomes identical to the separable case.

Soft vs Hard Margin SVMs

Even when the data can be linearly separated, we might benefit from using a soft margin, allowing us to get a much wider margin at a small cost (See figure 6.3-D). Using a Soft-Margin we can always obtain a solution, since the method is more robust to outliers. However, it requires us to guess the cost parameter, as opposed to the Hard-Margin method, which does not require any parameters.

6.7.4 Non Linear case

In some cases the data requires a more complex, non-linear separation (see figure 6.3-E). The non-linear case can be handled using techniques similar to what we discussed so far. Since finding a linear machine is not possible in the original space of the training set, we first map the training set to another Euclidean space with higher dimension (even infinite dimension). This higher-dimensional space is called the *feature space*, as opposed to the *input space* occupied by the training set (see figure 6.3-F). With an appropriately chosen feature space of sufficient dimensionality, any consistent training set can be made separable. However, translating the training set into a higher-dimensional space incurs a higher computational cost.

Suppose we first map the data to some other space H, using a mapping $\Phi : R^d \to H$. Then the SVM formulation becomes:

$$minimize \ \frac{1}{2}\|w\|^2 + C \sum_{i}^{l} \xi_i \tag{6.12}$$

$$\text{s.t. } y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i, \forall x_i, \ \xi_i \geq 0 \tag{6.13}$$

Data now appear as $\Phi(x_i)$. Weights are also mapped to the new space. However, if $\Phi(x_i)$ is very high dimensional, explicit mapping is very expensive. Therefore, we would like to solve the problem without explicitly mapping the data. The key idea is to notice that in the dual representation of the above problems - the training data appeared only in the form of dot products. Now suppose we first map the data to some other space H, using a mapping

$\Phi : R^d \rightarrow H$. Then the training algorithm would only depend on the data through dot products in H, i.e. on functions of the form $\Phi(x_i) \cdot \Phi(x_j)$. All we need in order to perform training in H is a function that satisfies $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, i.e., the image of the inner product of the data is the inner product of the images of the data. This type of function is called a *kernel function*. The kernel function is used in the higher dimension space as a dot product, so we do not need to explicitly map the data into the high-dimensional space. Classification can also be done without explicitly mapping the new instances to the higher dimension, we take advantage of the fact that $sgn(wx+b) = sgn(\sum_i \alpha_i y_i K(x_i, x) + b)$ where b solves $\alpha_j(y_j \sum_i \alpha_i y_i K(x_i, x_j) + b - 1) = 0$ for any j with $\alpha_j \neq 0$.

Examples of kernel functions are:

- $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$ - radial basis kernel.
- $K(x_i, x_j) = e^{\frac{x_i - x_j}{\sigma^2}}$ - gaussian kernel.
- $K(x_i, x_j) = (x_i \cdot x_j + 1)^k$ - polynomial kernel.

Bibliography

[1] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition, http://aya.technion.ac.il/karniel/cmcc/svm-tutorial.pdf. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[2] Anders Krogh J.Hertz and Richard G. Palmer. *Introduction to the theory of neural computation*. Perseus, 1991.

[3] Tom Mitchell. *Pattern Classification and Scene Analysis*. McGraw Hill, 1997.

[4] D.F. Morrison. *Multivariate Statistical Methods*. McGraw-Hill, 1990.

[5] Richard O.Duda and Peter E.Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons Inc., 1973.

[6] Yoav Freund Robert E. Schapire. A short introduction to boosting, http://www.site.uottawa.ca/ stan/csi5387/boost-tut-ppr.pdf. *Journal of Japanese Society for Artificial Intelligence*, 5(14):771–780, October 1999.

[7] V.N.Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1999.